A deontic logical framework for modelling product families

A deontic logical framework for modelling product families

Patrizia Asirelli, Maurice H. ter Beek, Stefania Gnesi
Istituto di Scienza e Tecnologie dell'Informazione
ISTI–CNR, Pisa, Italy
Email: {asirelli,terbeek,gnesi}@isti.cnr.it

Alessandro Fantechi
DSI, University of Florence, Italy
ISTI–CNR, Pisa, Italy
Email: fantechi@dsi.unifi.it

***Abstract*—We discuss the application of deontic logics to the modelling of variabilities in product family descriptions. Deontic logics make it possible to express concepts like permission and obligation, and hence promise a direct modelling of constraints over the products of a family. Indeed, we first show how feature models can be straightforwardly characterised by means of a deontic logic. We then study the deontic modelling of the behavioural variability in product families by defining a deontic extension of a behavioural logic. This allows both constraints over the products of a family and constraints over their behaviour to be expressed in a single framework: a novelty in the field. We discuss how model-checking tools could support formal verification in this framework, and we indicate some future research into that direction.**

I. Introduction

Modelling variability in product families has been the subject of extensive study in the literature on Software Product Lines, especially that concerning Feature modelling [3], [8], [15]. Variability modelling addresses how to define which features or components of a system are optional, alternative, or mandatory; formal methods are then developed to show that a product belongs to a family, or to derive instead a product from a family, by means of a proper selection of the features or components.

Modal Transition Systems (MTSs) have been proposed as a formal model for product families [12], [18], allowing one to embed in a single model the behaviour of a family of products that share the basic structure of states and transitions, transitions which can moreover be seen as mandatory or possible for the products of the family. In [10], we have pushed the MTS concept to a more general form, allowing more precise modelling of the different kinds of variability that can typically be found in the definition of a product family.

Recently, deontic logics [1], [22] have become popular in computer science for formalising descriptional and behavioural aspects of systems. This is mainly because they provide a natural way to formalise concepts like violation, obligation, permission, and prohibition. Intuitively, they permit one to distinguish between correct (normative) states and actions on the one hand and non-compliant states and actions on the other hand. This makes deontic logics a natural candidate for expressing the conformance of members of a family of products with respect to variability rules.

Such a conformance of products to a family concerns not only properties related to features, that in some sense can be considered static. Behavioural variability of the family has to be considered as well, i.e. how the products of a family differ in their ability to respond to events in time. These dynamic properties must also be verified for products to be member of such families. This is an aspect that the techniques focussing on feature models do not typically address.

Recently, a Propositional Deontic Logic (PDL) capable of expressing the permitted behaviour of a system has been proposed [5], [6]. This PDL combines the expression of permission and obligation with concepts from temporal logics.

In [2], we have laid the basis for the study of the application of deontic logics to the modelling of behavioural variability. We did this by showing the capability of a logic derived from PDL to finitely characterise the complete behaviour of a family of products. We have also shown in [2] that, given an MTS $\mathcal{M}$ representing a family, we are able to produce a deontic logic formula that is satisfied by all and only those products that can be derived from $\mathcal{M}$. This preliminary result has convinced us that this kind of deontic logics are a good candidate to express in a unique framework both behavioural variability aspects, by means of standard branching-time logical operators, and static constraints over the products of a family, which usually require a separate expression in a first-order logic (as can be seen in [3], [11], [19]), using deontic operators.

In the first part of this paper, we present a straightforward characterization of feature models by means of deontic logics. We then proceed with this direction of research by defining a novel deontic extension of a behavioural logic that allows both static constraints over the products of a family and constraints over their behaviour to be expressed in a single framework. This logic is given a semantic interpretation over MTSs for which a verification framework based on model-checking techniques could be implemented extending existing model checking tools.

A. Related Work

MTSs have been introduced in [12], [17], [18] to formally model and verify the behaviour of product families. We have extended MTSs in [10] to allow different notions

Funded by the Italian project D-ASAP (MIUR-PRIN 2007) and by the RSTL project XXL of the Italian National Research Council (CNR).

of behavioural variability to be modelled. An algebraic approach to behavioural modelling and verification of software product lines has been developed in [13], [14]. In [2] we showed how to finitely characterise MTSs by means of deontic logic formulae. To the best of our knowledge, the current paper presents a first attempt towards a modelling and verification framework capable of addressing both static and behavioural conformance of products of a family.

B. Outline

In Sect. II we present a simple running example that we will use throughout the paper. After a brief description of feature models in Sect. III, we discuss the use of deontic logic to characterise feature models and to verify static requirements of product families in Sect. IV. We then introduce the behavioural modelling of families by means of MTSs in Sect. V. In Sect. VI we introduce a deontic extension of an existing branching-time logic, which we apply to the running example in Sect. VII to show its expressivity and to discuss the verification of behavioural requirements of product families in Sect. VIII. We conclude the paper in Sect. IX with some ideas for future work.

II. Running Example: Coffee Machine Family

To illustrate the contribution of this paper we use a simple running example, namely a family of (simplified) coffee machines, for which we consider the following requirements:

1) A coffee machine is activated by a coin. The only accepted coins are the one euro coin (1€), exclusively for the European products and the one dollar coin (1$), exclusively for the US products;
2) After inserting a coin, the user has to choose whether or not (s)he wants sugar, by pressing one of two buttons, after which (s)he may select a beverage;
3) The choice of beverages (coffee, tea, cappuccino) varies between the products. However, delivering coffee is a must for every product of the family, while cappuccino is only offered by European products;
4) After delivering the appropriate beverage, optionally, a ringtone is rung. However, a ringtone must be rung whenever a cappuccino is delivered;
5) The machine returns to its idle state when the cup is taken by the user.

This list contains both static requirements, which identify the features that constitute the different products (see requirements 1, 3 and, partially, 4) and behavioural requirements, which describe the admitted sequences of operations (requirements 2, 5 and, partially, 4).

In the sequel, we will first distill the feature model of the above family and provide a formal representation in terms of deontic logic formulae. We will then show how the behavioural requirements of this family can be described using an MTS. Finally, we will show how to combine the two approaches by defining a deontic logical framework to check the satisfiability of both static and behavioural requirements over products that should belong to the family.

III. Feature Diagrams and Feature Models

Feature diagrams were introduced in [15] as a graphical *and/or* hierarchy of features; the features are represented as the nodes of a tree, with the product family being the root. Features come in several flavours; in this paper we consider the following features:

optional features	may be present in a product only if their parent is present;
mandatory features	are present in a product if and only if their parent is present;
alternative features	are a set of features among which one and only one is present in a product if their parent is present.

When additional constraints are added to a feature diagram, this results in a *feature model*. Also constraints come in several flavours; in this paper we consider the following constraints:

requires	is a unidirectional relation between two features indicating that the presence of one feature requires the presence of the other;
excludes	is a bidirectional relation between two features indicating that the presence of either feature is incompatible with the presence of the other.

An example of a feature model for the Coffee Machine of Sect. II is given in Fig. 1; the **requires** constraint obligates feature Ringtone to be present whenever Cappuccino is, while the **excludes** constraint prohibits features 1$ and Cappuccino to both be present in any product of this family of Coffee Machines. It is easy to see that this feature model satisfies the static requirements (1, 3 and, part of, 4) of our running example.

IV. Deontic Logic applied to Feature Models

Deontic logics are an active field of research for many years now. Many different deontic logic systems have been developed with a lot of success in the community [1], [22].

A. Deontic Logic: A First Glimpse

A deontic logic consists of the standard operators of propositional logic (i.e. negation ($\neg$), conjunction ($\wedge$), disjunction ($\vee$) and implication ($\Longrightarrow$)) augmented with deontic operators. In this paper, we consider two of the most classic deontic operators, namely *it is obligatory that* (O) and *it is permitted that* (P), which enjoy the duality property

$$P(\alpha) = \neg O(\neg\alpha),$$

i.e. something is permitted iff its negation is not obligatory.

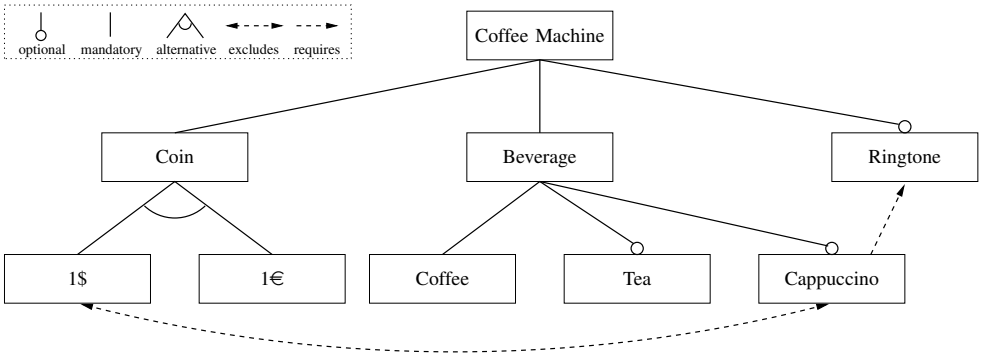

Figure 1. Feature model of the Coffee Machine family.

B. A Deontic Characterization of Feature Models

The way deontic logics formalise concepts such as violation, obligation, permission and prohibition is very useful for system specification, where these concepts arise naturally. In particular, deontic logics seem to be very useful to formalise product families specifications, since they allow one to capture the notions of optional and mandatory features.

The deontic characterization of a feature model builds a set of deontic formulae which, taken as a conjunction, precisely characterises a product family. We assume that a name of a feature A is used as the atomic proposition indicating that A is present.

The deontic characterization is constructed as follows:

- If A is a feature, and A_1 and A_2 are two subfeatures (possibly marked **alternative**, **optional** or **mandatory**), then add the formula

$$A \implies \Phi(A_1, A_2),$$

where $\Phi(A_1, A_2)$ is defined as:

$$\Phi(A_1, A_2) = (O(A_1) \vee O(A_2)) \wedge \neg(P(A_1) \wedge P(A_2))$$

if A_1 and A_2 are marked **alternative**,
and is otherwise defined as:

$$\Phi(A_1, A_2) = \phi(A_1) \wedge \phi(A_2),$$

in which A_i, for $i \in \{1, 2\}$, is defined as:

$$\phi(A_i) = \begin{cases} P(A_i) & \text{if } A_i \text{ is } \textbf{optional} \text{ and} \\ O(A_i) & \text{if } A_i \text{ is } \textbf{mandatory}. \end{cases}$$

Moreover, since the presence of the root feature is taken for granted, the premise of the implication related to that feature can be removed.[1]

- If A **requires** B, then add the formula

$$A \implies O(B)$$

- If A **excludes** B, then add the formula

$$(A \implies \neg P(B)) \wedge (B \implies \neg P(A))$$

Before applying this construction to our running example, we make two remarks. First, note that the **alternative** feature can also be defined in terms of the **excludes** feature: Marking subfeatures A_1 and A_2 **alternative** is the same as stating that A_1 and A_2 **exclude** each other.

The second remark concerns the following less common feature: **multiple optional** features are a set of features of which at least m and at most $n > m$, with $m, n \geq 0$, are present in a product if their parent is present. The deontic characterization of this feature is as follows, assuming that feature A has s subfeatures $A_1, \ldots, A_s$:

$$A \implies (\bigvee_{K} (\bigwedge_{i \in K} O(A_i) \wedge \bigwedge_{j \notin K} \neg P(A_j))),$$

with $K = \{\, S \subseteq 2^{\{1,\ldots,s\}} : m \leq |S| \leq n \,\}$.

If we apply the construction described above to our running example, then the conjunction of the following deontic formulae precisely characterises the feature model of Fig. 1. We refer to this conjunction as the *characteristic formula*.

$$O(\text{Coin}) \wedge O(\text{Beverage}) \wedge P(\text{Ringtone})$$

$$\text{Coin} \implies (O(1\$) \vee O(1€)) \wedge \neg(P(1\$) \wedge P(1€))$$
$$\text{Beverage} \implies O(\text{Coffee}) \wedge P(\text{Tea}) \wedge P(\text{Cappuccino})$$

$$\text{Cappuccino} \implies O(\text{Ringtone})$$
$$(1\$ \implies \neg P(\text{Cappuccino})) \wedge (\text{Cappuccino} \implies \neg P(1\$))$$

C. Verifying Static Requirements of Product Families

The deontic characterization of feature models described in this section is a way to provide a semantics to feature models. The deontic formulae expressing the features and the constraints between them can then be used to verify whether or not a certain product belongs to a specific family.

Given the above characterization of the coffee machine, let us suppose that we have defined two coffee machines with the following list of features:

$$\text{CM1} = \{\text{Coin}, 1€, \text{Beverage}, \text{Coffee}\}$$
$$\text{CM2} = \{\text{Coin}, 1€, \text{Beverage}, \text{Coffee}, \text{Cappuccino}\}$$

[1] By doing so, we tacitly do not deal with trivially inconsistent graphs in which the root is involved in an excludes relation with a feature.

It is easy to see that coffee machine CM1 belongs to the product family since it satisfies the characteristic formula of the feature model, whereas CM2 does not: it falsifies the constraint that a Cappuccino requires a Ringtone. This can be formally verified by interpreting these lists as a conjunction of axioms (each comma stands for a $\wedge$) that when added to the characteristic formula makes it either true or false, according to whether or not the product belongs to the family. For instance, CM2 does not belong to the product family because the addition (through conjunction) of

$$\text{Coin} \wedge 1€ \wedge \text{Beverage} \wedge \text{Coffee} \wedge \text{Cappuccino}$$

to the characteristic formula of the feature model makes the subformula Cappuccino $\implies O(\text{Ringtone})$ false, as a result of which the whole formula is false.

In general, the problem of finding a product that satisfies the deontic characterization of a feature model is reduced to that of finding a satisfying assignment to a set of boolean variables. Efficient SAT solvers, like Chaff [21], can therefore be used to address this kind of problems.

V. Behavioural Models for Product Families

The behavioural requirements given in Sect. II for our coffee machine family can be formally expressed by a Modal Transition System (MTS). Several variants of MTSs have been proposed in [10], [12], [18] with the aim of embedding in a single model the behaviour of a family of products that share the basic structure of states and transitions. This basic structure can be defined as a doubly-labelled transition system [9], which is an extension of an ordinary Labelled Transition System (LTS) obtained by labelling states with atomic propositions and transitions with actions.

Definition 1: A *doubly-labelled transition system* (L^2TS) is a sixtuple $(S, s_0, Act, \rightarrow, AP, L)$, in which

- S is a set of states;
- $s_0 \in S$ is the initial state;
- Act is a finite set of observable actions;
- $\rightarrow \subseteq S \times Act \times S$ is the transition relation; instead of $(s, \alpha, s') \in \rightarrow$ we will often write $s \xrightarrow{\alpha} s'$;
- AP is a set of atomic propositions;
- $L : S \longrightarrow 2^{AP}$ is a labelling function that associates a subset of AP to each state.

An MTS can now be defined as an L^2TS in which transitions are either required or possible, to reflect mandatory or optional transitions for the products of the family.

Definition 2: A *modal transition system* (MTS) is a septuple $(S, s_0, Act, \rightarrow_\Box, \rightarrow_\Diamond, AP, L)$ such that $(S, s_0, Act, \rightarrow_\Box \cup \rightarrow_\Diamond, AP, L)$ is an L^2TS. A MTS has two distinct transition relations: The *must* relation $\rightarrow_\Box$ expresses *required* transitions, while the *may* relation $\rightarrow_\Diamond$ expresses *possible* transitions. Note that, by definition, the may relation includes the must transition.

An example MTS is shown in Fig. 2; the solid arcs are must transitions, while the dashed arcs are may transitions. Its states are $\{0, 1, \ldots, 12\}$, with initial state 0, while its set of actions contains 1€, 1$, sugar, no_sugar, etc.

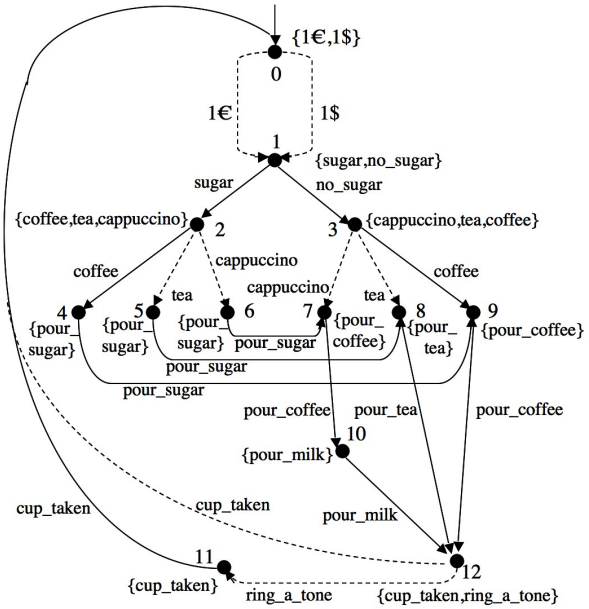

Figure 2. MTS modelling a product family.

Each state is labelled with the set of actions that label its outgoing (required and possible) transitions. Finally, must transitions $2 \xrightarrow{\text{coffee}}_\Box 4$ and $3 \xrightarrow{\text{coffee}}_\Box 9$ imply that delivering coffee is a must for every product of the family represented by this MTS. In fact, this MTS models the behavioural requirements given in Sect. II for our family of coffee machines.

Note that the MTS in Fig. 2 also models the static requirements concerning **optional** and **mandatory** features, through the use of may and must transitions. However, this MTS is not able to model three particular constraints listed among the requirements:

- actions 1€ and 1$ are exclusive (**alternative** features);
- a cappuccino is only offered by European products (**excludes** relation between features);
- a ringtone is rung whenever a cappuccino is delivered (**requires** relation between features).

We have seen that deontic logics can express also these characteristics, as represented by the Feature Diagram of Fig. 1. In order to define a unique framework in which to reason about behavioural as well as static requirements, in the following section we work on the integration of deontic operators within a temporal logic able to deal with MTSs.

VI. Towards a Deontic Logical Framework for Product Families

In order to define a unique logical framework in which to express both evolution in time and the variability of a product family, we define the temporal logic DHML based on the "Hennessy-Milner logic with Until" defined in [9], [16], which has been augmented with the Deontic *possibility* and *obligation* operators (in a style reminiscent of the logic PDL proposed in [5], [6]) and path operators from CTL [7]. DHML is a simpler variant of the logic proposed in [2].

A. DHML Logic: Syntax

DHML is a logic of state formulae (denoted by ϕ), in which a path quantifier prefixes an arbitrary path formula (denoted by π). We assume a set of atomic actions $Act = \{\alpha, \beta, \ldots\}$ and a set of atomic propositions $AP = \{p, q, \ldots\}$. From these two sets more complicated formulae can be built in the usual way, using the propositional and deontic operators described in Sect. IV or actions as well as the Hennessy-Milner modal, CTL path, and Until operators we describe next, together with their intuitive meaning:

- $[a]\,\phi$: for all next states reachable with a, ϕ holds;
- $E\,\pi$: there exists a path on which π holds;
- $A\,\pi$: on each of the possible paths π holds;
- $\phi\ U\ \phi'$: in the current or a future state ϕ' holds, while ϕ holds until that state (but not necessarily in that state).

Definition 3: The syntax of DHML is:

$$\begin{aligned} \phi \quad &::= \quad tt \mid p \mid \neg\phi \mid \phi \wedge \phi' \mid [\alpha]\phi \mid E\pi \mid A\pi \mid \\ &\qquad P(\alpha) \mid O(\alpha) \\ \pi \quad &::= \quad \phi\ U\ \phi' \end{aligned}$$

As usual, *ff* abbreviates $\neg tt$, $\phi \vee \phi'$ abbreviates $\neg(\neg\phi \wedge \neg\phi')$, $\phi \Longrightarrow \phi'$ abbreviates $\neg\phi \vee \phi'$, and $\langle\alpha\rangle\phi$ abbreviates $\neg[\alpha]\neg\phi$: there exists a next state reachable with a in which ϕ holds. Furthermore, $F\,\phi$ abbreviates $(tt\ U\,\phi)$: there exists a future state in which ϕ holds; and $G\,\phi$ abbreviates $\neg F\,(\neg\phi)$: in any future state ϕ holds. Finally, $EF\,\phi$ abbreviates $E\,(tt\ U\,\phi)$: there exists a path on which ϕ holds in a future state; and $AG\,\phi$ abbreviates $\neg EF\neg\phi$: ϕ holds in every state on every path.

An example of a well-formed formula in DHML is thus

$$[\alpha](P(\beta) \wedge (p \implies O(\gamma))),$$

which states that after the execution of the action α, the system is in a state where the action β is *permitted* (in the sense of the *may* transition) and if the proposition p holds then the action γ is *obligatory* (in the sense of the *must* transition).

B. DHML Logic: Semantics

The formal semantics of DHML is given below by means of an interpretation over the MTSs defined in Sect. V. To this purpose, we use a relation $P \subseteq S \times Act$ to denote which actions are permitted in which states, with the understanding that $P(s,\alpha)$ iff $\alpha \in L(s)$. We assume that $Act \subseteq AP$, i.e. all actions are atomic propositions.

Definition 4 (Semantics): The satisfaction relation $\models$ of DHML over an MTS $\mathcal{L} = (S, s_0, Act, \rightarrow_\Box, \rightarrow_\Diamond, AP \cup Act, L)$ is defined as follows:

- $s \models tt$ always holds;
- $s \models p$ iff $p \in L(s)$;
- $s \models \neg\phi$ iff not $s \models \phi$;
- $s \models \phi \wedge \phi'$ iff $s \models \phi$ and $s \models \phi'$;
- $s \models [\alpha]\phi$ iff $s \xrightarrow{\alpha}_\Diamond s'$, for some $s' \in S$, implies $s' \models \phi$;

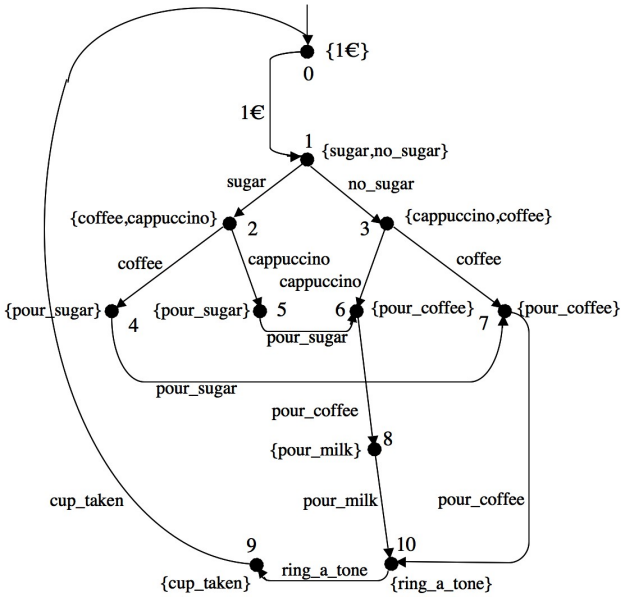

Figure 3. MTS of a European Coffee Machine.

- $s \models E\pi$ iff there exists a path σ starting in state s such that $\sigma \models \pi$;
- $s \models A\pi$ iff $\sigma \models \pi$ for all paths σ starting in state s;
- $s \models P(\alpha)$ iff $P(s,\alpha)$ holds;
- $s \models O(\alpha)$ iff $P(s,\alpha)$ holds and $\exists s' : s \xrightarrow{\alpha}_\Box s'$;
- $\sigma \models [\phi\ U\ \phi']$ iff there exists a state s_j, for some $j \geq 0$, on the path σ such that for all states s_k, with $j \leq k$, $s_k \models \phi'$ while for all states s_i, with $0 \leq i < j$, $s_i \models \phi$.

For the MTS in Fig. 2 we thus have, e.g., $0 \models 1€$ and $0 \models [1€](O(\text{sugar}))$ since $1 \models (O(\text{sugar}))$, which itself follows from the fact that sugar $\in L(1)$ and $1 \xrightarrow{\text{sugar}}_\Box 2$.

Note that notions of weak and strong permission are introduced in [5], [6] (and used to define a notion of obligation). The semantics of DHML can be extended in the following way to include a notion P_w of *weak permission*: $s \models P_\text{w}(\alpha)$ iff $P(s,\alpha)$ holds and $\exists s' : s \xrightarrow{\alpha}_\Diamond s'$.

Finally, we note that DHML differs from the classical modal μ-calculus [16], since the modal box operator of DHML is defined in terms of may transitions while the modal μ-calculus makes no distinction between must and may transitions in its semantic domain. For the same reason, also the weak permission operator cannot be expressed in the modal μ-calculus.

VII. Using DHML to Express Behavioural and Static Requirements of Product Families

We can now apply the DHML logic introduced in the previous section to our running example. We do this to illustrate the ability of DHML to express both static constraints over the products of a family and constraints over their behaviour.

DHML is able to complement the behavioural description of an MTS by expressing constraints over possible products of a family, that is, the static requirements that could not be expressed in the MTS:

- actions 1€ and 1$ are exclusive (**alternative** features):

$$((EF\ <1\$>\ \mathit{true}) \implies (AG\,\neg P(1€))) \wedge$$
$$((EF\ <1€>\ \mathit{true}) \implies (AG\,\neg P(1\$)))$$

- a cappuccino is only offered by European products (**excludes** relation between features):

$$((EF\ <\mathit{cappuccino}>\ \mathit{true}) \implies (AG\,\neg P(1\$))) \wedge$$
$$((EF\ <1\$>\ \mathit{true}) \implies (AG\,\neg P(\mathit{cappuccino})))$$

- a ringtone is rung whenever a cappuccino is delivered (**requires** relation between features):

$$(EF\ <\mathit{Cappuccino}>\ \mathit{true}) \implies (AF\,O(\mathit{ring\_a\_tone}))$$

The above expressions have been obtained by merging the static requirements represented by the pure deontic formulae given in Sect.IV for these requirements, with the behavioural relations among actions expressible by the temporal part of the logic. It is worthwhile making two remarks. First, note that we have used the same characterization of the **alternative** feature as the one given for the **excludes** feature.

Second, since **requires** is a static relation between features it does not imply any ordering among the related features, i.e. a coffee machine that rings a ringtone before producing a cappuccino cannot be excluded as a product of the family of coffee machines by verification of the above formula. Indeed, the correct ordering of actions is guaranteed by the MTS description of the family.

DHML is also able to express behavioural requirements over possible products of a family as temporal logic properties, such as:

1) It is possible to get a coffee with 1€:

$$[1€]\ EF\ <\mathit{coffee}>\ \mathit{true}$$

2) It is always possible to ask for sugar:

$$AF\ <\mathit{sugar}>\ \mathit{true}$$

3) It is not possible to get a beverage without inserting a coin:

$$AG\,(\neg(\mathit{coffee} \vee \mathit{tea} \vee \mathit{cappuccino})\ U\ (<1€>\ \mathit{true} \vee <1\$>\ \mathit{true}))$$

It is important to note, however, that the logical concept of *possibility* does not distinguish between the concepts *possibility for a user of the coffee machine to ask for sugar* and *possibility for a product of the family to include, among the other actions offered to the user, the action of asking a cappuccino*. To distinguish these two concepts of possibility, we need to resort to the deontic operators of DHML, using its capability to combine the expression of the concepts of permission and obligation with that of behavioural requirements.

VIII. Using DHML to Verify Behavioural and Static Requirements of Product Families

Another classic application of temporal logic is to verify that a model of a system satisfies properties given by logic formulae. Model checking is the most known automatic technique for verifying a system's correctness properties [7]. Such verification is exhaustive, i.e. all possible input combinations and states are taken into account, and a counterexample is usually generated in case a certain property does not hold. Correctness properties reflect typical (un)desired behaviour of the system under scrutiny. Formally, the problem of model checking can be stated as follows: given a desired property, expressed as a temporal logic formula ϕ, and a structure $\mathcal{M}$ with initial state 0, decide whether $\mathcal{M}, 0 \models \phi$, where $\models$ is the usual satisfaction relation. If $\mathcal{M}$ is finite, model checking thus reduces to a graph search.

We could use model checking to analyse the conformance of members of a family of products with respect to variability rules. To do so, we consider that a product is formally represented by a MTS in which only must (required) transitions appear. For instance, let us consider the product represented by the coffee machine defined by the MTS presented in Fig. 3. Such a MTS may have been generated starting from an independent high-level specification language such as, e.g., UML, and we may want to check that it belongs to the family, by checking the properties that we have defined to characterise the coffee machine family. It is easy to check that all the properties previously defined are satisfied by this MTS.

Moreover, if we take a few examples of properties expressed in DHML that are a mix of behavioural and deontic characteristics, then we are interested in checking their validity over the MTS presented in Fig. 3. If they turn out to be valid, then we can conclude that this product satisfies all the static (features) and behavioural requirements that the products derived from the family of coffee machines should satisfy according to the list of requirements given in Sect. II.

A first and simple example is the formula

$$EF\ O(\mathit{coffee}),$$

which must be read as *it is possible that eventually the product is obligated to deliver a coffee*, i.e. *there exists a sequence of actions that leads to a state in which there is a must transition labelled coffee*. Verifying this formula on this model of a product shows it is valid, because in state 2, e.g., there is a must transition labelled *coffee* ($2 \xrightarrow{\text{coffee}}_{\Box} 4$) and $\mathit{coffee} \in L(2)$.

Note that the presence of a may transition labelled *cappuccino* has no influence on the verification of this formula: To be valid in a state s, the obligation $O(\mathit{coffee})$ requires s to be labelled with *coffee* and the presence of an outgoing must transition from s labelled with *coffee*.

It is immediate that this formula implies the validity of

$$EF\ P(\mathit{coffee}).$$

Finally, since all paths at a certain point pass either state 2 or 3 and *coffee* $\in L(3)$ and $3 \xrightarrow{\text{coffee}}_{\Box} 7$, even the formula

$$AF\ O(\textit{coffee})$$

is valid: always eventually a coffee must be delivered.

In general, to perform verifications of this kind, we need a model checker able to check DHML formulae over models described as MTSs, with possible constraints expressed in DHML itself.

We are currently pursuing two different approaches to DHML model checking:

- We can exploit the relations between MTSs and L^2TSs in order to reuse the UMC model-checking engine [20]. UMC is an on-the-fly model checker that was originally designed for the efficient verification of UCTL logic [4], an action- and state-based branching-time temporal logic, over L^2TSs. The comparison of the expressiveness of UCTL and DHML still has to be studied, which means that enhancements to the model-checking engine to cover DHML deontic operators could be needed as well.
- Several model checkers employ SAT-solvers to implement the so-called *bounded model checking* approach, in order to efficiently address large state spaces. Using the same SAT-based engine for solving both the deontic issues related to the constraints on a family (as seen before) and the behavioural issues (employing bounded model-checking techniques) is hence a way of pursuing the scalability of verification of DHML properties to real-world cases, in which state spaces tend to increase beyond the capability of explicit model checkers.

Merging the two approaches with the aim of increased scalability and usability would then be a further step in the direction of the industrial application of the verification of behavioural requirements of product families.

IX. Conclusions and Future Work

In [2] we have shown how a deontic logic can express the variability of a product family by showing the capability of a deontic logic formula to finitely characterise a finite state MTS, a formal method proposed to capture the behavioural variability of a product family. In this paper, we have pursued this line of research. We have first shown how feature models can be straightforwardly characterised by means of a deontic logic. We have then defined DHML, a novel deontic extension of a Hennessy-Milner and CTL-like behavioural logic for product families that allows both static constraints over the products of a family and constraints over their behaviour to be expressed in a single framework. The semantic domain of this logic has been chosen such that a verification framework based on model-checking techniques is available.

The added value of the DHML logic we have introduced in this paper can thus be summarised as allowing the possibility of reasoning, in a unique framework, also on behavioural aspects of products of a family.

There are several aspects of our line of research that require a deeper understanding:

- how to express dependencies of variation points;
- the identification of classes of properties that, proved on family definitions, are preserved by all the products of the family;
- how quantitative properties can be evaluated, such as the number of possible different products of a given family;
- from a more pragmatic point of view, the study on scalability to real problems, and how the approach adapts to incremental family construction.

More importantly, it remains to study to what degree the complexity of the proposed deontic logic and verification framework can be hidden from the end user, or can be made more user friendly, in order to support developers in practice, since formal models such as MTSs are not directly usable as modelling framework. Nowadays, UML diagrams are often used as modelling paradigm and it could be very interesting to be able to apply to them the same formal reasoning we have presented here for MTSs and the deontic logic. Indeed, recently model-checking techniques for UML activity and state chart diagrams have been developed [4], exploiting the branching-time temporal logic UCTL. An extension of this framework by including deontic operators could be applied to verify the conformance of static and dynamic constraints of product derivations. This would allow to go into the direction of producing the family description itself already in a UML-like fashion, hence towards a better usability and acceptance within the industrial software product lines community.

References

[1] L. Åqvist, Deontic Logic. In D. Gabbay and F. Guenthner (Eds.): Handbook of Philosophical Logic (2nd Edition), Volume 8. Kluwer Academic, 2002, 147-264.

[2] P. Asirelli, M.H. ter Beek, A. Fantechi, and S. Gnesi, Deontic Logics for Modeling Behavioural Variability. In D. Benavides, A. Metzger, and U. Eisenecker (Eds.): *Proceedings of the Third International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'09)*, ICB Research Report 29, Universität Duisburg-Essen, 2009, 71–76.

[3] D.S. Batory, Feature Models, Grammars, and Propositional Formulas. In J.H. Obbink and K. Pohl (Eds.): *Proceedings Software Product Lines Conference (SPLC'05)*, LNCS 3714, 2005, 7–20.

[4] M.H. ter Beek, A. Fantechi, S. Gnesi and F. Mazzanti, An action/state-based model-checking approach for the analysis of communication protocols for Service-Oriented Applications. In S. Leue and P. Merino (Eds.): *Proceedings Formal Methods for Industrial Critical Systems (FMICS'07)*, LNCS 4916, Springer, 2008, 133–148.

[5] P.F. Castro and T.S.E. Maibaum, A Complete and Compact Propositional Deontic Logic. In C.B. Jones, Zh. Liu and J. Woodcock (Eds.): *International Colloquium Theoretical Aspects of Computing (ICTAC'07)*, LNCS 4711, Springer, 2007, 109–123.

[6] P.F. Castro and T.S.E. Maibaum, A Tableaux System for Deontic Action Logic. In R. van der Meyden and L. van der Torre (Eds.): *Proceedings Deontic Logic in Computer Science (DEON'08)*, LNCS 5076, Springer, 2008, 34–48.

[7] E.M Clarke, E.A. Emerson, and A.P. Sistla, Automatic Verification of Finite State Concurrent Systems using Temporal Logic Specifications. *ACM Transactions of Programming Languages and Systems* 8, 2 (1986), 244–263.

[8] K. Czarnecki and U.W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, 2000.

[9] R. De Nicola and F.W. Vaandrager, Three Logics for Branching Bisimulation. *Journal of the ACM* 42, 2 (1995), 458–487.

[10] A. Fantechi and S. Gnesi, Formal Modeling for Product Families Engineering. In *Proceedings Software Product Lines Conference (SPLC'08)*, IEEE, 2008, 193–202.

[11] A. Fantechi, S. Gnesi, G. Lami and E. Nesti, A Methodology for the Derivation and Verification of Use Cases for Product Lines. In R.L. Nord (Ed.): *Proceedings Software Product Lines Conference (SPLC'04)*, LNCS 3154, Springer, 2004, 255–265.

[12] D. Fischbein, S. Uchitel and V.A. Braberman, A Foundation for Behavioural Conformance in Software Product Line Architectures. In R.M. Hierons and H. Muccini (Eds.): *Proceedings Role of Software Architecture for Testing and Analysis (ROSATEA'06)*, ACM, 2006, 39–48.

[13] A. Gruler, M. Leucker and K.D. Scheidemann, Modeling and Model Checking Software Product Lines. In G. Barthe and F.S. de Boer (Eds.): *Proceedings Formal Methods for Open Object-Based Distributed Systems (FMOODS'08)*, LNCS 5051, Springer, 2008, 113–131.

[14] A. Gruler, M. Leucker and K.D. Scheidemann, Calculating and Modeling Common Parts of Software Product Lines. In: *Proceedings Software Product Lines Conference (SPLC'08)*, IEEE, 2008, 203–212.

[15] K. Kang, S. Choen, J. Hess, W. Novak and S. Peterson, Feature Oriented Domain Analysis (FODA) Feasibility Study. Technical Report SEI-90-TR-21, Carnegie Mellon University, Nov. 1990.

[16] K.G. Larsen, Proof Systems for Satisfiability in Hennessy-Milner Logic with Recursion, *Theoretical Computer Science* 72, 2-3, (1990), 265–288

[17] K.G. Larsen and B. Thomsen, Partial Specifications and Compositional Verification, *Theoretical Computer Science* 88, 1, (1991), 15–32

[18] K.G. Larsen, U. Nyman and A. Wąsowski, Modal I/O Automata for Interface and Product Line Theories. In R. De Nicola (Ed.): *Proceedings European Symposium on Programming Languages and Systems (ESOP'07)*, LNCS 4421, Springer, 2007, 64–79.

[19] M. Mannion and J. Camara, Theorem Proving for Product Line Model Verification. In F. van der Linden (Ed.): *Proceedings Software Product-Family Engineering (PFE'03)*, LNCS 3014, Springer, 2004, 211–224.

[20] F. Mazzanti, UMC model checker v3.6, April 2009. URL: `http://fmt.isti.cnr.it/umc`

[21] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang and S. Malik, Chaff: Engineering an Efficient SAT Solver. In: *Proceedings Design Automation Conference (DAC'01)*, ACM, 2001, 530–535.

[22] J.-J.Ch. Meyer and R.J. Wieringa (Eds.), *Deontic Logic in Computer Science: Normative System Specification*, Wiley, 1994.